Práctica 4. Exploración de grafos

Francisco Javier Molina Rojas
javier.molinarojas@alum.uca.es
Teléfono: 722528757
NIF: 45386606Q

9 de enero de 2023

1. Comente el funcionamiento del algoritmo y describa las estructuras necesarias para llevar a cabo su implementación.

   El Algoritmo sigue la estructura de un algoritmo A*. En el tenemos, dos vectores (Nopen y Nclosed) que simbolizan las listas de abierto y cerrado, un nodo inicial y un nodo objetivo entre otras variables. Estas especialmente son esenciales para el funcionamiento del algoritmo. Lo primero que realiza es meter al nodo inicial dentro de la lista de abiertos y una vez abierto todos sus adyacentes (y después de haber calculado el valor de las funciones g, h, f y el valor adicional de celda para cada uno de ellos), mete a dicho nodo en la lista de cerrados, para posteriormente abrir el próximo nodo con menor valor en la función f. Esto es debido a que es el nodo que más cerca está de la solución. Para realizar las ordenaciones, usamos un montículo en el vector que representa la lista de abiertos. Además, gracias a la función estaEnVect, podemos hacer que no se añadan a la lista de abiertos nodos que ya estén contenidos en esta y nodos que estén contenidos en la lista de cerrados. Una vez llegamos al nodo objetivo, volvemos atrás gracias a la referencia al nodo padre de cada nodo, consiguiendo así el camino hacia el nodo objetivo y la introducción de este dentro de la lista de posiciones path.

   También era necesario añadir la función comp para poder ordenar correctamente el montículo en función del valor de f para cada nodo.

2. Incluya a continuación el código fuente relevante del algoritmo.

Algoritmo De Encontrar el Camino

```
// ###### Config options ################

#define PRINT_PATHS 1

// ######################################

#define BUILDING_DEF_STRATEGY_LIB 1

#include "../simulador/Asedio.h"
#include "../simulador/Defense.h"

#ifdef PRINT_PATHS
#include "ppm.h"
#endif

using namespace Asedio;

Vector3 cellCenterToPosition(int i, int j, float cellWidth, float cellHeight){
    return Vector3((j * cellWidth) + cellWidth * 0.5f, (i * cellHeight) + cellHeight * 0.5f, 0);
}

bool comp(const AStarNode* A,const AStarNode* B)
{
        return A->F < B->F;
}

bool estaEnVect(AStarNode* A,std::vector<AStarNode*> V)
{
        bool esta = false;
        for(std::vector<AStarNode*>::iterator it = V.begin() ; it != V.end() ; it++)
        {
                if((*it)->position.x == A->position.x && (*it)->position.y == A->position.y)
                        esta = true;
        }
        return esta;
}

//anadiremos una coste adicional a aquellas posiciones que esten en el rango de una de las defensas
void DEF_LIB_EXPORTED calculateAdditionalCost(float** additionalCost
                    , int cellsWidth, int cellsHeight, float mapWidth, float mapHeight
```

```
                    , List<Object*> obstacles, List<Defense*> defenses)
{
        float cost = 0;

    float cellWidth = mapWidth / cellsWidth;
    float cellHeight = mapHeight / cellsHeight;

    for(int i = 0 ; i < cellsHeight ; ++i)
    {
        for(int j = 0 ; j < cellsWidth ; ++j) //recorrer casillas
        {
            Vector3 cellPosition = cellCenterToPosition(i, j, cellWidth, cellHeight); //obtener la posicion de la
                casilla
                        for(List<Defense*>::iterator it = defenses.begin() ; it != defenses.end() ; it++) //recorrer
                            defensas
                        {
                                //si la distancia entre la casilla y al defensa es menor que su rango, entonces
                                    estamos dentro de su rango
                                if(_sdistance(cellPosition,(*it)->position) <  (*it)->range)//le anadiremos un coste
                                    relacionado con las propiedades de la defensa
                                    cost += (0.5*(*it)->damage + 0.3*(*it)->health + 0.2*(*it)->attacksPerSecond
                                        ); //valoraremos mas el danio de la defensa, luego su salud y por
                                            ultimo su ataque por segundo
                        }
                        additionalCost[i][j] = cost; //asignamos coste a dicha casilla
                        cost = 0;       //reiniciamos el valor de coste para la siguiente
        }
    }
}

void DEF_LIB_EXPORTED calculatePath(AStarNode* originNode, AStarNode* targetNode
                    , int cellsWidth, int cellsHeight, float mapWidth, float mapHeight
                    , float** additionalCost, std::list<Vector3> &path)
{
    float totalCost = 0; //coste del camino dsde un punto a otro
    AStarNode* current = originNode; //nodo que usamos para movernos entre padres e hijos

        std::vector<AStarNode*>  Nopen,Nclosed; //vectores para almacenar los nodos abiertos y cerrados

        current->parent = NULL; //ponemos la referencia al padre del nodo origen a NULL ya que partimos de el

        Nopen.push_back(current); //anadimos el nodo origen al vector de abiertos

        std::make_heap(Nopen.begin(),Nopen.end()); //creamos un monticulo sobre este vector para obtener una
            ordenacion rapida

    while(!Nopen.empty()) //mientras que el vector de abiertos no este vacio:
    {

        while(current != targetNode && current != NULL && !Nopen.empty()) //mientras que el vector de abiertos no
            este vacio ni current sea (el nodo obj o null):
        {
                current = Nopen.front(); //Obtenemos el mejor nodo de la lista de abiertos
                std::pop_heap(Nopen.begin(),Nopen.end()); //traemos al final del vector el elemento a eliminar
                Nopen.pop_back(); //lo eliminamos

                        for (List<AStarNode*>::iterator it=current->adjacents.begin(); it != current->adjacents.end
                            (); ++it) //for para recorrer los adyacentes del nodo actual
                        {
                                if(!estaEnVect((*it),Nopen) && !estaEnVect((*it),Nclosed)) //si estos no estan en
                                    los vectores de nodos abiertos o cerrados
                                {
                                        (*it)->H = _sdistance((*it)->position,targetNode->position); //evaluamos la
                                            funcion heuristica (coste en llegar a la solucion)
                                        (*it)->G = _sdistance((*it)->position,originNode->position); //evaluamos la
                                            funcion g (coste en llegar a ese nodo especifico)
                                        (*it)->F = (*it)->H + (*it)->G + additionalCost[(int)((*it)->position.x /
                                            cellsWidth)][(int)((*it)->position.y / cellsHeight)]; //f valdra g + h
                                            y le anadimos un coste adicional determinado por la funcion
                                            calculateAdditionalCost
                                        (*it)->parent = current; //asignamos su respectivo padre
                                        Nopen.push_back((*it)); //lo metemos en la lista de abiertos
                                }
                        }

                        Nclosed.push_back(current); //una vez abierto todos sus posibles adyacentes, lo movemos a la
                            lista de cerrados
                        std::sort_heap(Nopen.begin(),Nopen.end()); //ordenamos el monticulo
        }

        if(current == targetNode) //si hemos llegado al nodo objetivo:
        {
                while(current != originNode)//recorremos el camino a la inversa para ver el coste de dicha solucion
                    (mientras que no lleguemos al nodo origen)
                {
                        totalCost += _sdistance(current->position,current->parent->position); //anadimos el coste de
                            llegar hacia el
                        path.push_front(current->position); //anadimos la posicion de dicho nodo al camino
                        current = current->parent; //cambiamos el nodo actual por por su padre
                        }
        }
    }
}
```

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de esta práctica confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.